Contents

[1. Support Vector Machine 3](#_Toc294179040)

[1.1. Definition 3](#_Toc294179041)

[1.2. Motivation 3](#_Toc294179042)

[1.3. Advantages of SVM 3](#_Toc294179043)

[1.4. Support Vector Machines Architecture 4](#_Toc294179044)

[1.4.1. SVM – a linear Classifier 4](#_Toc294179045)

[1.4.2. SVM – a Non-linear Classifier 5](#_Toc294179046)

[1.5. Algorithm of Support Vector Machine 5](#_Toc294179047)

[1.5.1. Linear SVM 6](#_Toc294179048)

[1.5.2. Non Linear SVMS 6](#_Toc294179049)

[1.5.3. Nonlinear SVM: the kernel trick 8](#_Toc294179050)

[1.6. Kernel function 9](#_Toc294179051)

[1.6.1. Inner-product kernel 9](#_Toc294179052)

[1.6.2. Mercer’s Theorem 10](#_Toc294179053)

[1.6.3. Examples of commonly-used kernel functions 11](#_Toc294179054)

[1.7. Support Vector Machine Application 11](#_Toc294179055)

[Text Categorization 11](#_Toc294179056)

[Cancer classification 12](#_Toc294179057)

[1.8. Drawback of Support Vector machine 12](#_Toc294179058)

[2. Radial Basis Function 14](#_Toc294179059)

[2.1. What is a radial basis function (RBF)? 14](#_Toc294179060)

[2.1.1. RBF Types 16](#_Toc294179061)

[2.1.2. Approximation 17](#_Toc294179062)

[2.1.3. Radial Basis Function Network 17](#_Toc294179063)

[2.1.4. Characteristics of RBF networks 19](#_Toc294179064)

[2.2. Motivation 19](#_Toc294179065)

[2.3. Advantages of RBF 20](#_Toc294179066)

[2.4. RBF Network Architecture 21](#_Toc294179067)

[2.4.1. Architecture Parameters 21](#_Toc294179068)

[2.4.2. Other Parameters 22](#_Toc294179069)

[2.4.3. Implementation 26](#_Toc294179070)

[2.4.4. Training RBF Networks 26](#_Toc294179071)

[2.5. Learning algorithm for RBF 27](#_Toc294179072)

[2.6. Applications 28](#_Toc294179073)

[2.7. Drawback of RBF 30](#_Toc294179074)

# Support Vector Machine

## Definition

A support vector machine (SVM) is a concept, for a set of related [supervised learning](http://en.wikipedia.org/wiki/Supervised_learning) methods that analyze data and recognize patterns; it is used for [classification](http://en.wikipedia.org/wiki/Classification_%28machine_learning%29) and [regression analysis](http://en.wikipedia.org/wiki/Regression_analysis). The standard SVM takes a set of input data and predicts, for each given input, which of two possible classes the input is a member of, which makes the SVM a non-[probabilistic](http://en.wikipedia.org/wiki/Probabilistic) [binary](http://en.wikipedia.org/wiki/Binary_classifier) [linear classifier](http://en.wikipedia.org/wiki/Linear_classifier).

Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

## Motivation

[Classifying data](http://en.wikipedia.org/wiki/Statistical_classification) is a common task in [machine learning](http://en.wikipedia.org/wiki/Machine_learning). Suppose some given data points each belong to one of two classes, and the goal is to decide which class a *new* data point will be in. In the case of support vector machines, a data point is viewed as a *p*-dimensional vector (a list of *p* numbers), and we want to know whether we can separate such points with a (*p* − 1)-dimensional [hyperplane](http://en.wikipedia.org/wiki/Hyperplane). This is called a [linear classifier](http://en.wikipedia.org/wiki/Linear_classifier). There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the [*maximum-margin hyperplane*](http://en.wikipedia.org/wiki/Maximum-margin_hyperplane) and the linear classifier it defines is known as a *maximum* [*margin classifier*](http://en.wikipedia.org/wiki/Margin_classifier), or - equivalently - the [*perceptron*](http://en.wikipedia.org/wiki/Perceptron) *of optimal stability.*

## Advantages of SVM

* Solves classification and regression problems.
* Efficiently computes (estimates) error rate and precision.
* Includes algorithm for approximately training large data.
* Handles many thousands of support vectors.
* Handles several hundred-thousands of training examples.
* Supports standard kernel functions and lets you define your own.
* The solution to an SVM is global and unique.
* Unlike ANN, the computational complexity of SVM does not depend on the dimensionality of the input space. ANN use empirical risk minimization, while SVM use structural risk minimization.
* The reason that SVM often outperform ANN in practice is that they deal with the biggest problem with ANN, SVM are less prone to overfiting.

## Support Vector Machines Architecture

### SVM – a linear Classifier

* We consider pattern classification tasks with training data

f(xi; yi), with xi element of R^n, and yi element of {−1; +1} is the class associated to input pattern xi.

* SVMs [3, 11] use a discriminant function with the following form:

Fa,b(x) = summation((aiyik(x,xi)+b )

Where SV, the set of support vectors, is a subset of the training patterns, and sign (f(x)) gives the class for any Pattern x. Parameters \_ and b are learned by the SVM algorithm, which also finds the set of support vectors, in

K

Linear Classifier Output = .x^~+b

F(*x)*

K

K

Linear classifier with

Parameters b and

Cost function being optimized

Mapped input vector x

Kernels with shared

K

Parameters

Support vectors:

Input vector x:

Figure 1 Architecture of SVM linear classifier

## **SVM – a Non-linear Classifier**

* We now have all the tools to construct nonlinear classifiers



* The parameters vi are computed as the solution of a quadratic programming problem.

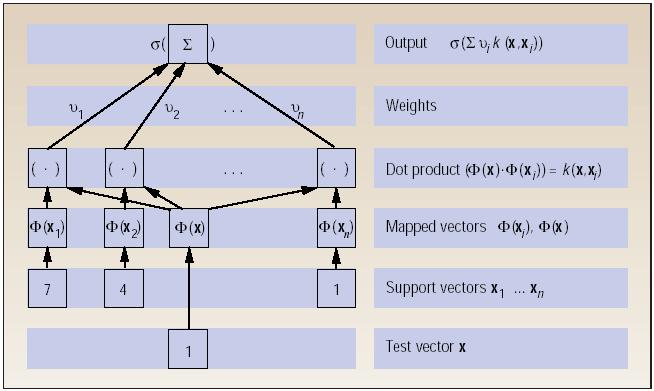


Figure 2 Architecture of SVM non linear classifier

## Algorithm of Support Vector Machine

* Choose a kernel function
* A kernel function is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:
* Choose a value for C
* Parameter C can be viewed as a way to control over fitting.
* Solve the quadratic programming problem
* Construct the discriminant function from the support vectors

here is the detail description:

### Linear SVM



* G(x) is a linear function:
* (Unit-length) normal vector of the hyper-plane:



* Given a set of data points:



Where



* With a scale transformation on both w and b, the above is equivalent to



Note:



* The margin width is:



### Non Linear SVMS

* Formulation:



Such that



* Formulation**:**



Such That



* Formulation**:**



Such That



And

* From KKT Condition, We Know:



* Thus, Only Support Vectors Have



* The solution has the form**:**



* The linear discriminant function is:



* Notice it relies on a dot product between the test point x and the support vectors xi
* Also keep in mind that solving the optimization problem involved computing the dot products xiTxj between all pairs of training points.

### Nonlinear SVM: the kernel trick

**Commonly-used kernel functions**



* + Linear kernel:



* + Polynomial kernel:
  + Gaussian (Radial-Basis Function (RBF) ) kernel:



* Sigmoid:
* In general, functions that satisfy *Mercer’s condition*

** can be kernel functions.

* Formulation: (Lagrangian Dual Problem)



Such that



* The solution of the discriminant function is



* The optimization technique is the same.

## Kernel function

### Inner-product kernel

Let x denote a vector drawn from the input space, assumed to be of dimension m0. Let {φj(*x)*}m1j=1 denote a set of nonlinear transformations from the input space to the feature space: m1 is the dimension of the feature space. It is assumed that φj(x) is defined a priori for all j. given such a set of nonlinear transformations; we may define a hyperplane acting as the decision surface as follows:



Where {wj} m1 j=1 denotes a set of linear weights connecting the feature space to the output space, and b is bias. We may simplify matters by writing



Where it is assumed that φ0(x) = 1 for all x, so that w0 denote the bias b. the above equation defines the decision surface computed in the feature space in terms of the linear weights of the machines. The quantity φj(x) represents the input supplied to the weight wj via the feature space. Define the vector

**φj(x)=[ φ0(x), φ1(x),……, φm1(x)]T**

Where, by definition, we have

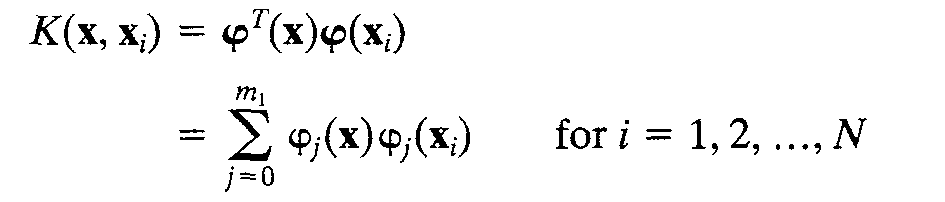
**φ0(x) = 1 for all x**

So here we can calculate the weight by using the following equation:



Where the feature vector **φ(x**) corresponds to the input patterns xj in the *ith* example. Therefore, substituting on the above equation, we may define the decision surface computed in the feature space as:



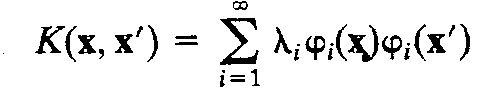
The term **φ(x**)T **φ(x**) represent the inner product of the two vectors induced in the feature space by the input vector x and the input pattern xj pertaining to the ith example. We may therefore introduce the inner-product kernel denoted by K(x,xi) and denoted by: 

From the above definition we immediately see that the inner-product kernel is symmetric function of its arguments, as shown by K(x,xi) = K(xi,x) for all i.

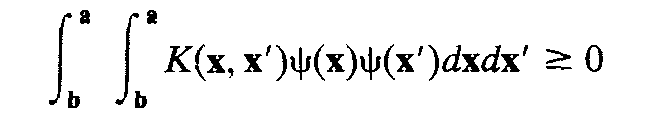
### Mercer’s Theorem

The expansion of inner-product kernel K(x, xi) is an important special case of Mercer’s theorem that arises in functional analysis.

Let K (x, x’) be a continuous symmetric kernel that is defined in the closed interval a<= x <=b and likewise for x’. The kernel K (x, x’) can be expanded in the series:



With the positive coefficient λi>0 for all i. For this expansion to be valid and for it to converge absolutely and uniformly, it is necessary and sufficient that the condition



The function φ(x) is called Eigen functions of the expansion and the number λi are called eigen values. The fact that all of the Eigen values are positive means that the kernel K(x,x’) is positive definite.

Mercer’s theorem only tells us whether or not a candidate kernel is actually an inner-product kernel in some space and therefore admissible for use in support vector machine. However, it says nothing about how to construct the function **φ(x**); we have to do those ourselves.

### Examples of commonly-used kernel functions



* + Linear kernel:
  + Polynomial kernel:



* + Gaussian (Radial-Basis Function (RBF) ) kernel:



* + Sigmoid:



## Support Vector Machine Application

SVM has been used successfully in many real-world problems

- Text (and hypertext) categorization

- Image classification

- Bioinformatics (Protein classification, Cancer classification)

- Hand-written character recognition

### Text Categorization

Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content.

- email filtering, web searching, sorting documents by topic, etc..

A document can be assigned to more than one category, so this can be viewed as a series of binary classification problems, one for each category.

A doc is represented by a vector indexed by a pre-fixed set or dictionary of terms

Values of an entry can be binary or weights:

figu151_1

Note that Doc x => **φ**(*x*)

The distance between two documents is φ(*x*)·φ(*z)*. *K*(*x,z*) = 〈φ(*x*)·φ(*z*) is a valid kernel, SVM can be used with *K*(*x,z*) for discrimination.

### Cancer classification

Support vector machine classification and validation of cancer tissue samples using microarray expression data.

DNA microarray experiments generating thousands of gene expression measurements are being used to gather information from tissue and cell samples regarding gene expression differences that will be useful in diagnosing disease. We have developed a new method to analyze this kind of data using support vector machines (SVMs). This analysis consists of both classification of the tissue samples, and an exploration of the data for mislabeled or questionable tissue results.

**E-learning**

Personalized and learner centered learning is receiving increasing importance due to increased learning rate.

SVMs stand out due to their better performance specially in handling large dimensions which text content do possess. Lecture material could be reprocessed to create a suitable feature space and then present the contents to the learner as per his need. This will save time and also avoid information overload.

## Drawback of Support Vector machine

In terms of running time, support vector machines are currently slower than other neural networks (e.g., multilayer perceptrons trained with the back-propagation algorithm) for a similar generalization performance.

If the number of features is much greater than the number of samples, the method is likely to give poor performances.

SVMs do not directly provide probability estimates, so these must be calculated using indirect techniques. In our case, these techniques imply conducting five-fold cross-validation, so performance can suffer.

It is sensitive to noise. A relatively small number of mislabeled examples can dramatically decrease the performance

* It only considers two classes
* How to do multi-class classification with SVM?

Answer:

* + 1. with output parity m, learn m SVM’s
    2. SVM 1 learns “Output==1” vs “Output != 1”

SVM 2 learns “Output==2” vs “Output! = 2”

:

SVM m learns “Output==m” vs “Output! = m”

* + 1. To predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

# Radial Basis Function

## What is a radial basis function (RBF)?

A radial basis function (RBF) is a real-valued function belonging to a class of functions that depend only on the Euclidean distance  of a domain point  from a variable domain point ξ and is of the form  or  where . These functions may be called globally supported or compactly supported depending on their supports, that is, whether they are defined on the whole domain or only on part of it. RBF may also have a shape parameter c in which case  is replaced by . RBFs have been used extensively in the context of multivariate interpolation with applications to geodesic approximation. Thereafter, RBFs have been widely applied in a number of fields such as neural networks, kinetic modeling and solution of differential and integral equations. Examples of the commonly used RBFs are given in Table 1.

Table 1: Some commonly used radial basis functions

|  |  |  |
| --- | --- | --- |
| Globally supported | Piecewise ploynomial | , n odd |
| Thin Plate Spline | , n even |
| Multiquadric (MQ) |  |
| Inverse multiqudric (IMQ) |  |
| Gaussian (GS) |  |
| Compactly supports | Wendland’s CSRBF |  |
|  |

The advantage of radial basis functions is that they involve a single independent variable, the distance r, regardless of the dimension of the problem A function f(x) that represents a physical problem, can be approximated using linear combinations of RBF centered at N number of points (RBF centers) distributed throughout the domain of computation, i.e.,

. Assuming that the values of are known at the N domain points such that, the unknown coefficients  can be obtained by satisfying the known values of f at the N domain points, i.e.  or in a matrix form Aα=h with entries of A given by, where =1,N. Once the coefficients “” are known, the RBF approximant can be used to find an approximation to the true solution of at all points in the computational domain and not just to the solution of at the node points considered in the solution of the problem. The above procedure can be illustrated by the following numerical example. Consider the function, where  and  to be interpolated by RBF, i.e.

, where .

Using the software Mathematica, which has the capability of solving systems of equations with symbolic and graphical representation of the solution, the results of the above collocation procedure yield the values of the coefficients  , k=1,N, and hence the complete form of the approximation. The results are given in Figures 1 which shows the absolute error of the approximation using 3x3, 5x5 and 9x9 node intensities, respectively. Figure 1 clearly shows that increasing the number of nodes will result in better RBF representation of the function. Furthermore, the error becomes high closer to the boundary decreasing dramatically from 50% for the 5x5 approximation to 0.5% for the 9x9 approximation.

Note that although the above RBF collocation representation was based on a group of predetermined points N in the problem domain, we have obtained a general representation that can be used to obtain the solution at any point in the domain that is not previously considered. Furthermore, we can generate symbolic expressions for the derivatives of the approximated function f(x) by direct differentiation. This represents an exceptional advantage for the RBF-based collocation method over other numerical methods.

### RBF Types

Commonly used types of radial basis functions include (writing r = \|\mathbf{x} - \mathbf{x}_i\|\;):

* [**Gaussian**](http://en.wikipedia.org/wiki/Gaussian_function)**:**

\phi(r) = e^{-(\varepsilon r)^2}\, 

* [**Multiquadric**](http://en.wikipedia.org/w/index.php?title=Multiquadric&action=edit&redlink=1)**:**

\phi(r) = \sqrt{1 + (\varepsilon r)^2} 

* [**Inverse Quadratic**](http://en.wikipedia.org/w/index.php?title=Inverse_Quadratic&action=edit&redlink=1)**:**

\phi(r) = \frac{1}{1+(\varepsilon r)^2} 

* [**Inverse Multiquadric**](http://en.wikipedia.org/w/index.php?title=Inverse_Multiquadric&action=edit&redlink=1)**:**

\phi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}} 

* [**Polyharmonic spline**](http://en.wikipedia.org/wiki/Polyharmonic_spline)**:**

\phi(r) = r^k,\; k=1,3,5,\dots

\phi(r) = r^k \ln(r),\; k=2,4,6,\dots 

* [**Thin plate spline**](http://en.wikipedia.org/wiki/Thin_plate_spline) **(a special polyharmonic spline):**

\phi(r) = r^2 \ln(r)\;

### Approximation

Radial basis functions are typically used to build up [function approximations](http://en.wikipedia.org/wiki/Function_approximation) of the form

y(\mathbf{x}) = \sum_{i=1}^N w_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|),

where the approximating function y(x) is represented as a sum of N radial basis functions, each associated with a different center xi, and weighted by an appropriate coefficient wi. The weights wi can be estimated using the matrix methods of [linear least squares](http://en.wikipedia.org/wiki/Linear_least_squares), because the approximating function is linear in the weights.

Approximation schemes of this kind have been particularly used in [time series prediction](http://en.wikipedia.org/wiki/Time_series_prediction) and [control](http://en.wikipedia.org/wiki/Control_theory) of [nonlinear systems](http://en.wikipedia.org/wiki/Nonlinear_systems) exhibiting sufficiently simple [chaotic](http://en.wikipedia.org/wiki/Chaos_theory) behaviour, 3D reconstruction in [computer graphics](http://en.wikipedia.org/wiki/Computer_graphics) (for example, [hierarchical RBF](http://en.wikipedia.org/wiki/Hierarchical_RBF)).

### Radial Basis Function Network

The *radial basis function (RBF) network* is a feed-forward neural network with a single layer of hidden units. Radial basis function networks are different from back propagation networks because they have only one layer of hidden units, and do not use the sigmoid activation function in the hidden layer unit. Instead, the radial basis function network has fixed-feature detectors in the hidden layer which use a specified basis function to detect and respond to localized portions of the input vector space. One advantage of radial basis networks over back propagation is that, if the input signal is non-stationary, the localized nature of the hidden layer response makes the networks less susceptible to "memory loss," or, as some would say, "weight loss."

A *radial basis function (RBF)* is a real-valued function whose value depends only on the distance from the [origin](http://en.wikipedia.org/wiki/Origin_%28mathematics%29), so that\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|); or alternatively on the distance from some other point c, called a center, so that\phi(\mathbf{x}, \mathbf{c}) = \phi(\|\mathbf{x}-\mathbf{c}\|). Any function φ that satisfies the property \phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)is a [radial function](http://en.wikipedia.org/wiki/Radial_function). The norm is usually [Euclidean distance](http://en.wikipedia.org/wiki/Euclidean_distance), although other [distance functions](http://en.wikipedia.org/wiki/Distance_function) are also possible. For example by using Lukaszyk-Karmowski metric, it is possible for some radial functions to avoid problems with [ill conditioning](http://en.wikipedia.org/wiki/Condition_number) of the matrix solved to determine coefficients wi (see below), since the \|\mathbf{x}\|is always greater than zero.

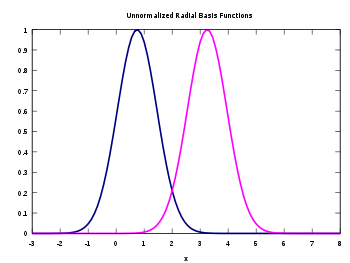
[](http://en.wikipedia.org/wiki/File:Unnormalized_radial_basis_functions.svg)

Figure 3 Un-normalized Radial Basis Function

Two un-normalized Gaussian radial basis functions in one input dimension. The basis function centers are located at x1=0.75 and x2=3.25.

The sum

y(\mathbf{x}) = \sum_{i=1}^N w_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|),

can also be interpreted as a rather simple single-layer type of [artificial neural network](http://en.wikipedia.org/wiki/Artificial_neural_network) called a [radial basis function network](http://en.wikipedia.org/wiki/Radial_basis_function_network), with the radial basis functions taking on the role of the activation functions of the network. It can be shown that any continuous function on a [compact](http://en.wikipedia.org/wiki/Compact_space) interval can in principle be interpolated with arbitrary accuracy by a sum of this form, if a sufficiently large number N of radial basis functions is used.

The approximant y(x) is differentiable with respect to the weights wi. The weights could thus be learned using any of the standard iterative methods for neural networks.

### Characteristics of RBF networks

In conclusion, it can be stated that RBF networks:

Provide a global approximation to the target function, represented by a linear combination of many local kernel functions;

* This can be viewed as the smooth linear combination of piece-wise (local) non-linear functions – that is the best function chosen for a particular range of input data;
* Training is faster than backpropagation networks since it is done in two steps;
* The accuracy of the solution is highly dependent on the range and quality of data;

it is an so-called eager method, but used an idea of local approximation as in lazy methods such as k-nearest neighbors. (Lazy methods that do not build a model on the basis of all available data, as ANN do, but rather “lazily” process new instances as they appear and the result is generated on the basis of training examples).

## Motivation

Radial Basis Functions (RBFs) have motivations such as:

* When Input rather than target is noisy
* There is a basis function centered on every data point
* For better performance than Sigmoid function

- Some classification problems

- Function interpolation

- Much of the inspiration for RBF networks has come from traditional statistical pattern classification techniques

## Advantages of RBF

The use of "global neural networks'' (as the back propagation neural network) and ``clustering neural networks'' (as the radial basis function neural network) leads each other to different advantages and disadvantages. The combination of the desirable features to those two neural ways of computation is achieved by the use of Modular Neural Networks (MNN). In addition, a considerable advantage can emerge from the use of such a MNN: an interpretable and relevant neural representation about the plant's behavior. This very desirable feature for function approximation and especially for control problems, is what lake other neural models. This feature is so important that we introduce it as a way to differentiate MNN between other local computation models. However, to enable a systematic use of MNN three steps have to be achieved. First of all, the task has to be decomposed into subtasks, then the neural modules have to be properly organized considering the subtasks and finally a way of communication inter-modules has to be integrated in the whole architecture. We achieved a study of the main modular applications according to those steps. This study leads to the main fact that a systematic use of MNN depends on the type of task considered. The clustering networks and especially the Local Model Networks can be seen as MNN in the frame of classification or recognition problems. The Euclidean distance criterion that they apply to cluster the input space leads to a relevant decomposition according to the properties of those tasks. But, it is irrelevant to apply such a criteria in case of function approximation problems. As spatial clustering seems to be the only existing decomposing method, therefore, an ``ad hoc'' decomposition and organization of the architecture is achieved in case of function approximation. So, to improve the systematic use of MNN in the framework of function approximation it is now essential to conceive a method of relevant task decomposition.

Neural networks require a lot of training to understand the model of a plant or a process. Issues such as learning speed, stability, and weight convergence remain as areas of research and comparison of many training algorithms. The application of neural networks to control interior permanent magnet synchronous motor using direct torque control (DTC) is discussed. A neural network is used to emulate the state selector of the DTC. The neural networks used are the back-propagation and radial basis function. To reduce the training patterns and increase the execution speed of the training process, the inputs of switching table are converted to digital signals, i.e., one bit represent the flux error, one bit the torque error, and three bits the region of stator flux. Computer simulations of the motor and neural-network system using the two approaches are presented and compared. Discussions about the back-propagation and radial basis function as the most promising training techniques are presented, giving its advantages and disadvantages. The system using back-propagation and radial basis function networks controller has quick parallel speed and high torque response.

An advantage of RBF networks is that when the basis functions are appropriately fixed, the network outputs become linear functions of the output layer weights.

It is also reported that RBF type ANNs are capable of universal approximations too.

For hypothetical situations, it also has been demonstrated that RBF type networks learn faster than MLP networks.

In RBF networks, one major advantage is that if the number of input variables is not too high, so the learning is much faster than in other types of networks. However, the required number of the hidden units increases geometrically with the number of the input variables. It becomes practically impossible to use this network for a large number of input variables.

The advantage of radial basis functions is that they involve a single independent variable, the distance r, regardless of the dimension of the problem A function f(x) that represents a physical problem, can be approximated using linear combinations of RBF centered at N number of points (RBF centers) distributed throughout the domain of computation, i.e.,.

## RBF Network Architecture

### Architecture Parameters

When creating a radial basis network, you must specify the following architecture parameters:

**Auto Center**

Controls whether the first layer weights are learned or set. A value of 1 indicates the center weights are determined using self-organizing learning. A value of 0 indicates the center weights must be explicitly set.

**Number of inputs**

Sets the number of units allocated for the input layer. This must be an integer value greater than or equal to 1.

**Number of basis units in the hidden layer**

Sets the number of basis units allocated for the hidden layer. This must be an integer value greater than or equal to 0.

**Number of outputs**

Sets the number of units allocated for the output layer. This must be an integer value greater than or equal to 1.

### Other Parameters

**Learn Rate**

Controls how much the weights are changed during a weight update. The larger the value, the more the weights are changed. This must be a real value between 0.0 and 10.0.

**All Widths**

Controls the width or selectivity of the selected radial basis function. If set to a non-zero value, all of the hidden units will use the parameter value; smaller values denote smaller, more focused selection. A zero value is not currently supported; for some implementations, a 0 value signals that each hidden unit takes a unique width parameter from the Widths array.

**Basis Function**

Selects which radial basis function is used for the activation function in the hidden units. Three functions are currently supported:

0. Gaussian = exp (- *v*\*\*2 / 2 width\*\*2)

1. Thin Plate Spline = *v*\*\*2 \* log (*v*)

2. Multi-quadratic = sqrt (v\*\*2 + width\*\*2)

where v is the Euclidian Norm, the distance between the Input vector and the hidden unit Center, calculated as the square root of the sum of the squared element differences:   
sqrt(&Sigma.(Input[i] - Center[i])\*\*2)

**Normalized**

Controls whether the hidden unit activations are normalized so that they sum to 1. If Normalized = 1, then all hidden unit values are summed and then divided by the sum. If Normalized = 0, then the hidden unit activation values are set to the value returned by the selected radial basis function. Normalization tends to force a single result rather than hedging between several responses.

**Momentum**

Controls how much the weights are changed during a weight update by factoring in previous weight updates. It acts as a smoothing parameter that reduces oscillation and helps attain convergence. This must be a real value between 0.0 and 1.0, a typical value for momentum is 0.5.

**Last RMS Error**

Indicates the root-mean-square (RMS) of the error for a single training pattern. When the number of output units is *n*, the formula is:   
sqrt ((&Sigma.&epsilon.\*\*2)/*n*)

**Ave RMS Error**

Indicates the average RMS error of the patterns in the previous epoch.

**Tolerance**

Sets the value for the acceptable difference between the desired output value and the actual output value. This must be a real value between 0.0 and 1.0. For example, if your training data set contains expected values of 0 and 1 and the tolerance is set to 0.1 (the default), then the average pattern error goes to 0 when all of the outputs are within 0.1 of the desired values.

**Epoch Updates**

Controls whether the network weights are updated after every pattern presentation (False) or only after a complete training epoch (True).

**Last Num Bad Outputs**

Indicates the number of output units that are out of the specified tolerance for a single training pattern.

**Bad Pattern Ratio**

Indicates the number of patterns in the previous epoch which have errors above tolerance divided by the total number of patterns.

**Max RMS Error**

Indicates the maximum RMS error of the patterns in the previous epoch.

The basic architecture for a RBF is a 3-layer network, as shown in Fig.

The input layer is simply a fan-out layer and does no processing.

The second or hidden layer performs a non-linear mapping from the input space into a (usually) higher dimensional space in which the patterns become linearly separable.



Figure 4 RBF Network architecture

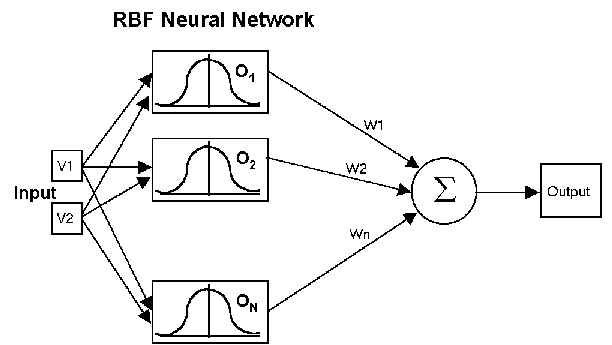


Figure 5 RBF Network architecture

RBF networks have three layers:

1. **Input layer** – There is one neuron in the input layer for each predictor variable. In the case of categorical variables, N-1 neurons are used where N is the number of categories. The input neurons (or processing before the input layer) standardize the range of the values by subtracting the median and dividing by the interquartile range. The input neurons then feed the values to each of the neurons in the hidden layer.
2. **Hidden layer** – This layer has a variable number of neurons (the optimal number is determined by the training process). Each neuron consists of a radial basis function centered on a point with as many dimensions as there are predictor variables. The spread (radius) of the RBF function may be different for each dimension. The centers and spreads are determined by the training process. When presented with the x vector of input values from the input layer, a hidden neuron computes the Euclidean distance of the test case from the neuron’s center point and then applies the RBF kernel function to this distance using the spread values. The resulting value is passed to the the summation layer.
3. **Summation layer** – The value coming out of a neuron in the hidden layer is multiplied by a weight associated with the neuron (W1, W2, ...,Wn in this figure) and passed to the summation which adds up the weighted values and presents this sum as the output of the network. Not shown in this figure is a bias value of 1.0 that is multiplied by a weight W0 and fed into the summation layer. For classification problems, there is one output (and a separate set of weights and summation unit) for each target category. The value output for a category is the probability that the case being evaluated has that category.

### Implementation

The radial basis function as implemented has a single hidden layer of units which can use one of three basis functions: Gaussian, thin plate spline, or multi-quadratic. The basis centers or weight vectors are learned during an initial stage of self-organized learning, currently fixed at 15 epochs. Once the basis vectors are set, the output layer weights are adjusted using back propagation.

### Training RBF Networks

The following parameters are determined by the training process:

1. The number of neurons in the hidden layer.
2. The coordinates of the center of each hidden-layer RBF function.
3. The radius (spread) of each RBF function in each dimension.
4. The weights applied to the RBF function outputs as they are passed to the summation layer.

Various methods have been used to train RBF networks. One approach first uses K-means clustering to find cluster centers which are then used as the centers for the RBF functions. However, K-means clustering is a computationally intensive procedure, and it often does not generate the optimal number of centers. Another approach is to use a random subset of the training points as the centers.

DTREG uses a training algorithm developed by Sheng Chen, Xia Hong and Chris J. Harris. This algorithm uses an evolutionary approach to determine the optimal center points and spreads for each neuron. It also determines when to stop adding neurons to the network by monitoring the estimated leave-one-out (LOO) error and terminating when the LOO error beings to increase due to over fitting.

The computation of the optimal weights between the neurons in the hidden layer and the summation layer is done using ridge regression. An iterative procedure developed by Mark Orr (Orr, 1966) is used to compute the optimal regularization Lambda parameter that minimizes generalized cross-validation (GCV) error.

## Learning algorithm for RBF

Given the number of the hidden nodes (centers) J is chosen, the learning algorithm is formulated as follows:

1. Find the positions of centers {wj}. This can be done by the following procedure:
2. • Choose randomly J instances xj and use them as the positions of the centers {wj}
3. • All the remainder of the instances (training patterns) is assigned to a class j of the closest centre wj, and the locations of each center are calculated again using for example k-nearest neighbor method.
4. • The above steps are repeated until the locations of the centers stop changing.
5. Calculate the output from each hidden neuron as a function of a radial distance from the input vector to the radial center. Calculated distance between the center and the input vector is passed through a non-linear mapping Gaussian function.
6. Weights {bjk} for the output layer are calculated using methodologies as in MLP, using backpropagation.

Where bjk – the weight on the connection from the hidden node j to the output node k, yj - the output from the hidden node j

1. Calculate the error between the network’s output and the target output and if the error of the network’s output is more than the desired limit then the number of the hidden units are changed and all the steps are repeated again.

## Applications

Radial basis function networks have been employed in many different problems even though they have not been used as often as multi-layer perceptrons. However, in the literature, the number of applications covered by RBFNs is quite high. Some of the major application areas for RBFNs can be listed as follows:

* finite element or [spectral methods](http://www.scholarpedia.org/article/Spectral_methods) for the solution of partial differential equations
* neural networks with radial basis functions, and machine learning
* approximations on spheres
* statistical approximations, where positive definite kernels are very important
* geophysical research
* and many engineering applications
* [Reconstruction and Representation of 3D Objects with Radial Basis Functions](http://www.farfieldtechnology.com/products/toolbox/theory/siggraph01.pdf)
* [Surface Interpolation with Radial Basis Functions for Medical Imaging](http://www.farfieldtechnology.com/products/toolbox/theory/ieeecran.ps.gz)
* [Smooth surface reconstruction from noisy range data](http://www.farfieldtechnology.com/products/toolbox/theory/graphite03.pdf)
* Speech recognition
* Image processing
* Time-series analysis
* Adaptive equalization
* Radar point source location
* Medical diagnosis
* Process faults detection
* Pattern recognition

Radial basis functions are presented as a practical solution to the problem of interpolating incomplete surfaces derived from three-dimensional (3D) medical graphics. The specific application considered is the design of cranial implants for the repair of defects, usually holes, in the skull.

## Drawback of RBF

* As can be seen, the RBF networks are employed mostly in classification problems. The main classification problem involving RBF networks is the pattern recognition problem.
* Time-series analysis is the second most common application area for RBF networks.
* Although the RBF is quick to train, when training is finished and it is being used it is slower than a MLP, so where speed is a factor a MLP may be more appropriate.
* Disadvantage of Adaptive RBF: Takes longer to run.